Steven M. Hoffberg

From: Steven M. Hoffberg [steve@hoffberg.org]
Sent: Thursday, November 18, 2004 12:19 PM
To: 'Nguyen, Nga'
Subject: 09/599,163 service_db.c

```
/* ---------------------------------------------------------------------------
 * ---------------------------------------------------------------------------
 * write-side access to the provider name databases
 * ---------------------------------------------------------------------------
 * ---------------------------------------------------------------------------
 */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <strings.h>
#include <bstring.h>
#include <ctype.h>
#include <sys/types.h>

#include "tvsd.h"
#include "tvs_util.h"
#include "gdbm.h"

#ifndef PRIVATE
#define PRIVATE static
#define PUBLIC
#endif /* PRIVATE */

#define DEF_BLOCK 512
#define SEPARATOR "|"

extern gdbm_error gdbm_errno;

struct pm_xlate  publisher;
struct tvs_xlate tvsserver;

/* ---------------------------------------------------------------------------
 * remove_blanks - "purify" strings (by removing trailing blanks)
 * ---------------------------------------------------------------------------
 */

PRIVATE void
remove_blanks(char *p)
{
   if(!p) return;
```

```
  while (*p) p++;
  --p;
  while (isspace(*p)) {
    *p = '\0';
    --p;
  }
}

/* ---------------------------------------------------------------------
 * read_entry - read in the next valid input line from PM or TVS file
 * ---------------------------------------------------------------------
 */

#ifdef SINGLE_SITE
int
read_entry(FILE *fp, int *id, char *nm, char *hp, int *sin)
#else
read_entry(FILE *fp, int *id, char *nm, char *hp)
#endif
{
  char buffer[256];

  do {
    if(fgets(buffer, 256, fp) == (char *) NULL) return 0;

    /* remove blank lines and \n termination */

    if((buffer[0] == '#') || (buffer[0] == '\n')) continue;
    buffer[strlen(buffer)-1] = '\0';

#ifdef SINGLE_SITE
    sscanf(buffer, "%d | %[^|] | %[^|] | %d", id, nm, hp, sin);
#else
    sscanf(buffer, "%d | %[^|] | %s", id, nm, hp);
#endif
    if (!*id || !nm[0] || !hp[0]) {
      fprintf(stderr, "bad line in input file: %s\n", buffer);
    }
    else
      break;
  } while(!feof(fp));

  /* remove trailing blanks from names */

  remove_blanks(nm);
  remove_blanks(hp);

  return 1;
}
```

```
/* ---------------------------------------------------------------------------
 * handle_fatal - handler for gross gdbm violations
 * ---------------------------------------------------------------------------
 */

PRIVATE void
handle_fatal(char *msg)
{
  sprintf(msgString,"fatal: error %d msg: %s", gdbm_errno, msg);
  LogMsg(LOG_ERR, msgString);

  return;
}

/* ---------------------------------------------------------------------------
 * Open the PM name database
 * ---------------------------------------------------------------------------
 */

PUBLIC GDBM_FILE
open_pm_db(int mode)
{
  GDBM_FILE dbf;

  if (mode <= 0) mode = GDBM_READER;
  dbf = gdbm_open("pm_name_by_id", DEF_BLOCK, mode, 00644, handle_fatal);
  return dbf;
}

/* ---------------------------------------------------------------------------
 * marshall/unmarshall PM from pm_xlate form to character string
 * ---------------------------------------------------------------------------
 */

#ifdef SINGLE_SITE
PUBLIC CONTENTS
pm_marshall (unsigned long pm_id, char *name, char *hostport, int single)
#else
PUBLIC CONTENTS
pm_marshall (unsigned long pm_id, char *name, char *hostport)
#endif
{
  CONTENTS cont;
  char *p, *q;
  int len;

#ifdef SINGLE_SITE
  len = sizeof(unsigned long) + strlen (name) + 1 + strlen (hostport) + 1
      + sizeof(int);
#else
  len = sizeof(unsigned long) + strlen (name) + 1 + strlen (hostport) + 1:
```

```
#endif

  p = q = malloc (len);
  if (!p) {
    cont.dsize = 0;
    cont.dptr = name;
    return cont;
  }

  *((unsigned long *) p) = pm_id;
  p += sizeof(unsigned long);
  strcpy (p, name);
  p += strlen(name) + 1;
  strcpy (p, hostport);
#ifdef SINGLE_SITE
  p += strlen(hostport) + 1;
  *((int *) p)  = single;
#endif

  cont.dptr = q;
  cont.dsize = len;
  return cont;
}

PUBLIC CONTENTS
pm_marshall_PM_XLATE (PM_XLATE pe)
{
#ifdef SINGLE_SITE
  return pm_marshall (pe->pm_id, pe->name, pe->hostport, pe->single);
#else
  return pm_marshall (pe->pm_id, pe->name, pe->hostport);
#endif
}

PUBLIC PM_XLATE
pm_unmarshall (CONTENTS cont)
{
  char *p;
  PM_XLATE pe;

  pe = (PM_XLATE) malloc (sizeof (struct pm_xlate));

  if (!pe)
    return NULL;

  p = cont.dptr;

  pe->pm_id = *((unsigned long *) p);
  p += sizeof(unsigned long);
  pe->name = strdup(p);
  p += strlen(p) + 1;
```

```
  pe->hostport = strdup(p);
#ifdef SINGLE_SITE
  p += strlen(p) + 1;
  pe->single = *((int *) p);
#endif

  return pe;
}

/* ---------------------------------------------------------------------------
 * add a PM to the PM name database
 * ---------------------------------------------------------------------------
 */

PRIVATE int
#ifdef SINGLE_SITE
pm_add_change(GDBM_FILE dbf, unsigned long pm_id, char *name, char *hostport,
              int single, int flags)
#else
pm_add_change(GDBM_FILE dbf, unsigned long pm_id, char *name, char *hostport,
              int flags)
#endif
{
  KEY key;
  CONTENTS cont;
  unsigned long pid;

  pid = pm_id;
  key.dptr = (char *) &pid;
  key.dsize = sizeof(unsigned long);

#ifdef SINGLE_SITE
  cont = pm_marshall(pm_id, name, hostport, single);
#else
  cont = pm_marshall(pm_id, name, hostport);
#endif

  if (!cont.dsize)
    return 0;

  return gdbm_store(dbf, key, cont, flags);
}

/* ---------------------------------------------------------------------------
 * add a PM to the xlator database
 * ---------------------------------------------------------------------------
 */

#ifdef SINGLE_SITE
PUBLIC int
pm_add(GDBM_FILE dbf, unsigned long pm_id, char *name, char *hostport, int s)
```

```
{
  int err;

  err = pm_add_change (dbf, pm_id, name, hostport, s, GDBM_INSERT);
  if (err != 0) {
    sprintf(msgString, "pm_add error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
  }
  return 0;
}
#else
PUBLIC int
pm_add(GDBM_FILE dbf, unsigned long pm_id, char *name, char *hostport)
{
  int err;

  err = pm_add_change (dbf, pm_id, name, hostport, GDBM_INSERT);
  if (err != 0) {
    sprintf(msgString, "pm_add error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
  }
  return 0;
}
#endif
#endif

/* ----------------------------------------------------------------------
 * replace a PM to the xlator database
 * ----------------------------------------------------------------------
 */

#ifdef SINGLE_SITE
PUBLIC int
pm_replace(GDBM_FILE dbf, unsigned long pm_id, char *name, char *hostport, int s)
{
  int err;

  err = pm_add_change (dbf, pm_id, name, hostport, s, GDBM_REPLACE);
  if (err != 0) {
    sprintf(msgString, "pm_replace error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
  }
  return 0;
}
#else
PUBLIC int
pm_replace(GDBM_FILE dbf, unsigned long pm_id, char *name, char *hostport)
{
```

```
  int err;

  err = pm_add_change (dbf, pm_id, name, hostport, GDBM_REPLACE);
  if (err != 0) {
    sprintf(msgString, "pm_replace error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
  }
  return 0;
}
#endif

/* ---------------------------------------------------------------------------
 * change a pM in the PM xlator database
 * ---------------------------------------------------------------------------
 */


#ifdef SINGLE_SITE
PUBLIC int
pm_change(GDBM_FILE dbf, unsigned long pm_id, char *name, char *hostport, int s)
{
  KEY key;
  CONTENTS cont;
  unsigned long id;
  int err;

  id = pm_id;
  key.dptr = (char *) &id;
  key.dsize = sizeof(unsigned long);

  cont = gdbm_fetch(dbf, key);
  if (cont.dptr == (char *) NULL) {
    sprintf(msgString, "pm_change error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
  }

  err = pm_add_change (dbf, pm_id, name, hostport, s, GDBM_REPLACE);
  if (err != 0) {
    sprintf(msgString, "pm_change error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
  }

  return 0;
}
#else
PUBLIC int
pm_change(GDBM_FILE dbf, unsigned long pm_id, char *name, char *hostport, int s)
{
```

```
  KEY key;
  CONTENTS cont;
  unsigned long id;
  int err;

  id = pm_id;
  key.dptr = (char *) &id;
  key.dsize = sizeof(unsigned long);

  cont = gdbm_fetch(dbf, key);
  if (cont.dptr == (char *) NULL) {
    sprintf(msgString, "pm_change error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
  }

  err = pm_add_change (dbf, pm_id, name, hostport, s, GDBM_REPLACE);
  if (err != 0) {
    sprintf(msgString, "pm_change error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
  }

  return 0;
}
#endif

/* ---------------------------------------------------------------------------
 * pm exists in the PM database
 * ---------------------------------------------------------------------------
 */

PUBLIC int
exists_pm(GDBM_FILE dbf, unsigned long p_id)
{
  KEY key;
  unsigned long id;

  id = p_id;
  key.dptr = (char *) &id;
  key.dsize = sizeof(unsigned long);

  return gdbm_exists(dbf, key);
}

/* ---------------------------------------------------------------------------
 * delete_pm from the PM database
 * ---------------------------------------------------------------------------
 */

PUBLIC int
```

```
pm_delete(GDBM_FILE dbf, unsigned long p_id)
{
  KEY key;
  int err;
  unsigned long id;

  id = p_id;
  key.dptr = (char *) &id;
  key.dsize = sizeof(unsigned long);

  err = gdbm_delete(dbf, key);
  if (err != 0) {
    sprintf(msgString, "pm_delete error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
  }

  return 0;
}

/* ---------------------------------------------------------------------------
 * pm_show in the PM database
 * ---------------------------------------------------------------------------
 */

PUBLIC void
pm_show(GDBM_FILE dbf)
{
  KEY key;
  CONTENTS cont;
  PM_XLATE pe;

  key = gdbm_firstkey(dbf);
  if (!key.dptr) return;

  printf("PM database:\n");

  do {
    cont = gdbm_fetch(dbf, key);
    if (cont.dptr) {
        pe = pm_unmarshall (cont);
#ifdef SINGLE_SITE
        printf("pm: %ld is \"%s\" on host \"%s\" (SINGLE)\n",
                pe->pm_id, pe->name, pe->hostport);
#else
        printf("pm: %ld is \"%s\" on host \"%s\"\n",
                pe->pm_id, pe->name, pe->hostport);
#endif
    }
    key = gdbm_nextkey(dbf, key);
  } while (key.dptr);
```

```
}

/* ---------------------------------------------------------------------------
 * pm_dump
 * ---------------------------------------------------------------------------
 */

PUBLIC void
pm_dump(GDBM_FILE dbf, FILE *out)
{
  KEY key;
  CONTENTS cont;
  PM_XLATE pe;

  key = gdbm_firstkey(dbf);
  if (!key.dptr) return;

  do {
    cont = gdbm_fetch(dbf, key);
    if (cont.dptr) {
        pe = pm_unmarshall (cont);
#ifdef SINGLE_SITE
        fprintf(out, "%ld | %s | %s | %d\n",
                pe->pm_id, pe->name, pe->hostport, pe->single);
#else
        fprintf(out, "%ld | %s | %s\n",
                pe->pm_id, pe->name, pe->hostport);
#endif
    }
    key = gdbm_nextkey(dbf, key);
  } while (key.dptr);
}

/* ---------------------------------------------------------------------------
 * pm_read -
 * ---------------------------------------------------------------------------
 */

PUBLIC PM_XLATE
pm_read(FILE *fp)
{
  int id;
  char nm[128], hp[128];
  struct pm_xlate *pub;

#ifdef SINGLE_SITE
  int sin;
  if (! read_entry(fp, &id, nm, hp, &sin))
#else
  if (! read_entry(fp, &id, nm, hp))
#endif
```

```
    return (struct pm_xlate *) NULL;

  pub = (struct pm_xlate *) malloc(sizeof(struct pm_xlate));
  if (!pub) return (struct pm_xlate *) NULL;

  pub->pm_id = id;
  pub->name = (char *) malloc(strlen(nm) +1);
  if (pub->name)
    strcpy(pub->name, nm);
  else
    return (struct pm_xlate *) NULL;

  pub->hostport = (char *) malloc(strlen(hp) +1);
  if (pub->hostport)
    strcpy(pub->hostport, hp);
  else
    return (struct pm_xlate *) NULL;

#ifdef SINGLE_SITE
  pub->single = sin;
  fprintf(stderr, "%ld | %s | %s (s)\n",  pub->pm_id, pub->name, pub->hostport);
#else
  fprintf(stderr, "%ld | %s | %s\n",  pub->pm_id, pub->name, pub->hostport);
#endif
  return pub;
}

/* ----------------------------------------------------------------------------
 * close the PM db file
 * ----------------------------------------------------------------------------
 */

PUBLIC int
close_pm_db(GDBM_FILE dbf)
{
  gdbm_sync(dbf);
  gdbm_close(dbf);
  return 1;
}

/* ----------------------------------------------------------------------------
 * open the TVS server database
 * ----------------------------------------------------------------------------
 */

PUBLIC GDBM_FILE
open_tvs_db(int mode)
{
  GDBM_FILE dbf;
  if (mode <= 0) mode = GDBM_READER;
  dbf  = gdbm_open("tvs_name_by_id", DEF_BLOCK, mode, 00644, handle_fatal);
```

```
  return dbf;
}

/* ----------------------------------------------------------------------------
 * marshall/unmarshall TVS server database entry
 * ----------------------------------------------------------------------------
 */

PUBLIC CONTENTS
tvs_marshall(unsigned short tvss_id, char *name, char *hostport)
{
  CONTENTS cont;
  char *p, *q;
  int len;

  len = sizeof(unsigned short) + strlen (name) + 1 + strlen (hostport) + 1;

  p = q = malloc (len);
  if (!p) {
    cont.dsize = 0;
    cont.dptr = p;
    return cont;
  }

  *((unsigned short *) p) = tvss_id;
  p += sizeof(unsigned short);
  strcpy (p, name);
  p += strlen(name) + 1;
  strcpy (p, hostport);

  cont.dptr = q;
  cont.dsize = len;
  return cont;
}

PUBLIC CONTENTS
tvs_marshall_TVS_XLATE (TVS_XLATE te)
{
  return tvs_marshall (te->tvss_id, te->name, te->hostport);
}

PUBLIC TVS_XLATE
tvs_unmarshall (CONTENTS cont)
{
  char *p;
  TVS_XLATE te;

  te = (TVS_XLATE) malloc (sizeof (struct tvs_xlate));

  if (!te)
    return NULL;
```

```
  p = cont.dptr;

  te->tvss_id = *((unsigned short *) p);
  p += sizeof(unsigned short);
  te->name = strdup(p);
  p += strlen(p) + 1;
  te->hostport = strdup(p);

  return te;
}

/* ---------------------------------------------------------------------------
 * add/change a TVS server to the TVS name database
 * ---------------------------------------------------------------------------
 */

PRIVATE int
tvs_add_change(GDBM_FILE dbf, unsigned short tvs_id, char *name,
               char *hostport, int flags)
{
  KEY key;
  CONTENTS cont;
  unsigned short tid;

  tid = tvs_id;
  key.dptr = (char *) &tid;
  key.dsize = sizeof(unsigned short);

  cont = tvs_marshall (tvs_id, name, hostport);
  if (!cont.dsize)
    return 0;

  return gdbm_store(dbf, key, cont, flags);
}

/* ---------------------------------------------------------------------------
 * add a TVS server to the TVS name database
 * ---------------------------------------------------------------------------
 */

PUBLIC int
tvs_add(GDBM_FILE dbf, unsigned short tvs_id, char *name, char *hostport)
{
  int err;

  err = tvs_add_change (dbf, tvs_id, name, hostport, GDBM_INSERT);
  if (err != 0) {
    sprintf(msgString, "tvs_add error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
```

```
  }

  return 0;
}

/* ---------------------------------------------------------------------------
 * replace a TVS server in the TVS name database
 * ---------------------------------------------------------------------------
 */

PUBLIC int
tvs_replace(GDBM_FILE dbf, unsigned short tvs_id, char *name, char *hostport)
{
  int err;

  err = tvs_add_change (dbf, tvs_id, name, hostport, GDBM_REPLACE);
  if (err != 0) {
    sprintf(msgString, "tvs_replace error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
  }

  return 0;
}

/* ---------------------------------------------------------------------------
 * change a TVS entry from TVS database
 * ---------------------------------------------------------------------------
 */

PUBLIC int
tvs_change(GDBM_FILE dbf, unsigned short tvs_id, char *name, char *hostport)
{
  KEY key;
  CONTENTS cont;
  unsigned short id;
  int err;

  id = tvs_id;
  key.dptr = (char *) &id;
  key.dsize = sizeof(unsigned short);

  cont = gdbm_fetch(dbf, key);
  if (cont.dptr == (char *) NULL) {
    sprintf(msgString, "tvs_change error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
  }

  err = tvs_add_change(dbf, tvs_id, name, hostport, GDBM_REPLACE);
  if (err != 0) {
```

```
    sprintf(msgString, "tvs_change error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
  }

  return 0;
}

/* ---------------------------------------------------------------------------
 * tvs exists in the TVS database
 * ---------------------------------------------------------------------------
 */

PUBLIC int
exists_tvs(GDBM_FILE dbf, unsigned short tvs_id)
{
  KEY key;
  unsigned short id;

  id = tvs_id;
  key.dptr = (char *) &id;
  key.dsize = sizeof(unsigned short);

  return gdbm_exists(dbf, key);
}

/* ---------------------------------------------------------------------------
 * delete_tvs from the PM database
 * ---------------------------------------------------------------------------
 */

PUBLIC int
tvs_delete(GDBM_FILE dbf, unsigned short t_id)
{
  KEY key;
  int err;
  unsigned short id;

  id = t_id;
  key.dptr = (char *) &id;
  key.dsize = sizeof(unsigned short);

  err = gdbm_delete(dbf, key);
  if (err != 0) {
    sprintf(msgString, "tvs_delete error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return -1;
  }

  return 0;
}
```

```
/* ---------------------------------------------------------------------------
 * show_tvss in the PM database
 * ---------------------------------------------------------------------------
 */

PUBLIC void
tvs_show(GDBM_FILE dbf)
{
   KEY key;
   CONTENTS cont;
   TVS_XLATE pe;

   key = gdbm_firstkey(dbf);
   if (!key.dptr) return;

   printf("PM database:\n");

   do {
      cont = gdbm_fetch(dbf, key);
      if (cont.dptr) {
            pe = tvs_unmarshall (cont);
            printf("pm: %d is \"%s\" on host \"%s\"\n",
                  pe->tvss_id, pe->name, pe->hostport);
      }
      key = gdbm_nextkey(dbf, key);
   } while (key.dptr);
}

/* ---------------------------------------------------------------------------
 * tvs_dump
 * ---------------------------------------------------------------------------
 */

PUBLIC void
tvs_dump(GDBM_FILE dbf, FILE *out)
{
   KEY key;
   CONTENTS cont;
   TVS_XLATE pe;

   key = gdbm_firstkey(dbf);
   if (!key.dptr) return;

   do {
      cont = gdbm_fetch(dbf, key);
      if (cont.dptr) {
            pe = tvs_unmarshall(cont);
            fprintf(out, "%ld | %s | %s\n",
                  pe->tvss_id, pe->name, pe->hostport);
      }
```

```
    key = gdbm_nextkey(dbf, key);
  } while (key.dptr);
}

/* ---------------------------------------------------------------------------
 * tvs_read -
 * ---------------------------------------------------------------------------
 */

PUBLIC TVS_XLATE
tvs_read(FILE *fp)
{
  int id;
  char nm[128], hp[128];
  struct tvs_xlate *tvss;

  if (! read_entry(fp, &id, nm, hp))
    return (struct tvs_xlate *) NULL;

  tvss = (struct tvs_xlate *) malloc(sizeof(struct tvs_xlate));
  if (!tvss) return (struct tvs_xlate *) NULL;

  tvss->tvss_id = (unsigned short) id;
  tvss->name = (char *) malloc(strlen(nm) +1);
  if (tvss->name)
    strcpy(tvss->name, nm);
  else
    return (struct tvs_xlate *) NULL;

  tvss->hostport = (char *) malloc(strlen(hp) +1);
  if (tvss->hostport)
    strcpy(tvss->hostport, hp);
  else
    return (struct tvs_xlate *) NULL;

  fprintf(stderr, "tvs_read: %ld | %s | %s\n",
          tvss->tvss_id, tvss->name, tvss->hostport);
  return tvss;
}

/* ---------------------------------------------------------------------------
 * close the TVS db file
 * ---------------------------------------------------------------------------
 */

PUBLIC int
close_tvs_db(GDBM_FILE dbf)
{
  gdbm_sync(dbf);
  gdbm_close(dbf);
  return 1;
```

```
}

/* ---------------------------------------------------------------------------
 * ---------------------------------------------------------------------------
 * read only access to the translator databases
 * ---------------------------------------------------------------------------
 * ---------------------------------------------------------------------------
 */

PRIVATE PM_XLATE
get_pm_xlate(GDBM_FILE dbf, unsigned long p_id)
{
  KEY key;
  CONTENTS cont;

  unsigned long id;

  id = p_id;
  key.dptr = (char *) &id;
  key.dsize = sizeof(unsigned long);

  cont = gdbm_fetch(dbf, key);
  if (cont.dptr == (char *) NULL) {
    sprintf(msgString, "get_pm_xlate error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return (PM_XLATE) NULL;
  }

  return pm_unmarshall(cont);
}

PUBLIC char *
get_pm_name(GDBM_FILE dbf, unsigned long p_id)
{
  PM_XLATE entry;

  entry = get_pm_xlate(dbf, p_id);
  if (!entry)
   return (char *) NULL;
  else
   return entry->name;
}

PUBLIC char *
get_pm_hostport(GDBM_FILE dbf, unsigned long p_id)
{
  PM_XLATE entry;

  entry = get_pm_xlate(dbf, p_id);
  if (!entry)
   return (char *) NULL;
```

```
  else
   return entry->hostport;
}
#ifdef SINGLE_SITE
PUBLIC int
pm_is_single_site(GDBM_FILE dbf, unsigned long pm_id)
{
  PM_XLATE entry;

  entry = get_pm_xlate(dbf, p_id);
  if (!entry)
   return -1;
  else
   return entry->single
}
#endif /* SINGLE_SITE */

PRIVATE TVS_XLATE
get_tvs_xlate(GDBM_FILE dbf, unsigned short t_id)
{
  KEY key;
  CONTENTS cont;

  unsigned short id;

  id = t_id;
  key.dptr = (char *) &id;
  key.dsize = sizeof(unsigned short);

  cont = gdbm_fetch(dbf, key);
  if (cont.dptr == (char *) NULL) {
    sprintf(msgString, "get_tvs_xlate error: %s\n", gdbm_strerror(gdbm_errno));
    LogMsg(LOG_ERR, msgString);
    return (TVS_XLATE) NULL;
  }

  return tvs_unmarshall (cont);
}

PUBLIC char *
get_tvs_name(GDBM_FILE dbf, unsigned short t_id)
{
  TVS_XLATE entry;

  entry = get_tvs_xlate(dbf, t_id);
  if (!entry)
   return (char *) NULL;
  else
   return entry->name;
}
```

```
PUBLIC char *
get_tvs_hostport(GDBM_FILE dbf, unsigned short t_id)
{
  TVS_XLATE entry;

  entry = get_tvs_xlate(dbf, t_id);
  if (!entry)
   return (char *) NULL;
  else
   return entry->hostport;
}
```

Very truly yours,

Steven M. Hoffberg
Milde & Hoffberg, LLP
Suite 460
10 Bank Street
White Plains, NY 10606
(914) 949-3100 tel.
(914) 949-3416 fax
steve@hoffberg.org
www.hoffberg.org

Confidentiality Notice: This message, and any attachments thereto, may contain confidential information which is legally privileged. The information is intended only for the use of the intended recipient, generally the individual or entity named above. If you believe you are not the intended recipient, or in the event that this document is received in error, or misdirected, you are requested to immediately inform the sender by reply e-mail at Steve@Hoffberg.org and destroy all copies of the e-mail file and attachments. You are hereby notified that any disclosure, copying, distribution or use of any information contained in this transmission other than by the intended recipient is strictly prohibited.